Recursion and Backtracking

They are one the most powerful technique in world of programming. They can solve almost any question ever given in competitive programming but problem they pose are their time complexities. We discussed recursion before but now we are here to discuss backtracking.

What is BackTracking?

Let's say you are on road which is diverging into three parts. One goes to exit while other goes to deadend. So one of the way is to go through one path, if you reach deadend then backtrack your steps to the initial position.

Let us understand it with the famous **N-Queens**

Problem Statement:
Given $N \times N$ chessboard, place N queens on it such that no queen can cut each other.

Facts to use:

1. Since queen can move horizontally, vertically and diagonally, we can only place one queen in one row and one column
2. Queen cannot be placed at a position where it cuts other queens

Strategy or Approach:
Mark each position to which previously placed queens can move as true on chess board, and for placing next queen try to place queen on a place which is false. If no then we will try to place this queen elsewhere.

Coding phase
STEP 1: We will create list of 2D array(vector here) which will tell if any queen can currently attack this position or not
STEP 2: We will try to place our queen at a point which is not currently attack and mark all the points this queen can attack if placed here
STEP 3: We will then ask our function to place next queen in i^{th} row
STEP 4: If next step is possible, then we are okay and placed all our queens

Solution

```cpp
#include<bits/stdc++.h>
using namespace std;
vector<vector<int> > arr;
int n;

bool NQueen(int i, vector<vector<int> > attacked){
    if(i == n)return true;
    for(int j = 0; j < n; ++j){
        if(attacked[i][j] == 0){
            vector<vector<int> > this_move = attacked;
            arr[i][j] = 1;
            for(int k = 0; k < n; ++k){
                this_move[i][k] = 1;
                this_move[k][j] = 1;
            }
            for(int k = 0; i+k < n && j+k < n; ++k){
                this_move[i+k][j+k] = 1;
            }
            for(int k = 0; i+k >= 0 && j+k >= 0; --k){
                this_move[i+k][j+k] = 1;
            }
            for(int k = 0; i+k < n && j-k >= 0; ++k){
                this_move[i+k][j-k] = 1;
            }
            for(int k = 0; i-k >= 0 && j+k < n; ++k){
                this_move[i-k][j+k] = 1;
            }
            if(NQueen(i+1,this_move)){
                return true;
            }
            arr[i][j] = 0;
        }
    }
    return false;
}

int main(){
    cin >> n;
    arr = vector<vector<int> > (n, vector<int>(n,0));
    vector<vector<int> > attacked (n, vector<int>(n,0));
    if(NQueen(0,attacked)){
        for(auto i: arr){
            for(auto j: i){
                cout << j << " ";
            }
            cout << endl;
        }
    }else{
        cout << "Not possible";
    }
return 0;
}
```

Backtracking is an expensive procedure (exponential time complexity), but when constraints are small we can use it to solve really complex constraints.

TIP: Whenever time complexities are exponential, constants should be of order ≤ 20

Questions for backtracking:

1. N-Queens
2. A Tryst with Chess
3. Curiosity has no limit
4. Pumping Water(Divide and Conquer)
5. Its Confidential
6. Simran and Stairs
7. MYSTERY
8. XSquare and Two Strings
9. Rajan and Odd Frequency Numbers
10. Subset AND